Planetary Lander Using Phasic Policy Gradient Algorithm

*P*riyanka Prabhath
200599373
Supervisor: Dr. Angadh Nanjangud
MSc Artificial Intelligence

***Abstract*—This paper presents a method to solve the rocket landing problem by using the phasic policy gradient algorithm. Phasic policy gradient improves sample efficiency while incorporating the benefits of proximal policy optimization which is the current standard algorithm used for solving different problems in the continuous state and action space domain. It is seen that with improved sample efficiency, there is a better convergence occurring which leads to the agent learning faster. Space exploration had become important for human kind. One of the crucial problems that need to be solved is landing an exploration object on an extra terrestrial surface. Since terrains of the surfaces are unknown, having the object plan and map out the surroundings is needed.**

I. Introduction

The main reasons for the advancement of human kind is their curiosity and the need for exploration. From looking up to the sky in bewilderment to landing man on the moon to missions to Mars, the human race has come a long way in space exploration with the different challenges faced which has led to many technological advances that we use today like the GPS, accurate weather predictions etc. It has also helped us study the different phenomenon experienced within earth with a different perspective. Today's space exploration is directed towards obtaining resources, finding planets similar to the earth and colonization on other planetary bodies. [13]

The first autonomous system was launched in 1998 by Deep Space 1 - the first fully autonomous spacecraft experiment. This showed that autonomous systems were more resilient and feasible as it reduces the operational costs needed. However it took some time for the adoption of autonomous systems on future space missions. Automating the simple tasks like maintenance of systems, conducting menial experiments or other objectives can take a load off the people making it much more cost effective that manually doing the jobs. The other reason for automation is the need for spacecrafts to be self - reliant due to the time difference between the earth and the point in space where the object is situated at. The time taken to communicate with the earth and the plane the object is situated at might be longer than the time needed to make a decision which makes it difficult for the space craft. With autonomous systems, complex issues that are near impossible for humans to solve can also be handled making it easier. [14]

Initial automated missions were rule-based. The space craft was treated as a state machine which consisted of different set behaviours and had transitions defined between the different modes. Thus all the systems were designed with pre-defined mode sequences or were based on some pre-launch criteria making the requirement of computing power low. But these rigid systems were prone to break apart when there were drastic changes to the different parameters such hardware failures or a new objective coming in place. When it comes to multiple competing mission objectives, there was little to no self support in order to integrate them. There was a need for human intervention in order to make a decision on the mission to support. [15]

In order to mitigate the problems faced by rule - based autonomy, optimization-based autonomy are a class of algorithms that are built on a framework that optimizes the constraints and metrics put up by the spacecraft's hardware and trajectory like the images, communication time etc. This leads to the need of higher computation power and thus increasing the cost. At the same time, this gives the system more autonomy thereby the spacecraft has better chances of surviving the unknow areas with limited human interaction. [15]

With emerging techniques coming up, the cost of mission development has reduced with higher increase of mission returns. With the emergence of ML and AI, the approaches to decision making has become more robust. Advanced ML techniques have been seen to make a huge impact on how space missions are conducted with the decision making process more flexible and adaptable to the continuously changing parameters and environment. [15]

Reinforcement Learning has been crucial in the development of autonomy in different fields especially in aerospace. Reinforcement learning was developed form different areas of study such as statistics, psycology, control theory and so on. By the late 80s, there was a wide acceptance of RL technologies opening up a vast field of research. RL algorithms has the ability to learn without any prior knowledge of its goal or environment and only depends on the feedback or reward signal provided by the environment. The agent learns and improves its strategy of obtaining the accurate actions needed for the environment with the feedback obtained from the environment. [16]

For space exploratory missions, landing on the surface of the planetary body is the primary objective that needs to be solved. The space craft needs to decent vertically as well as softly on the surface [17]. One of the initial methods investigated on the

bodies of the Moon and Mars is the Gravity - Turn descent towards the surface[29] [17]. It involves orienting the velocity vector and the lander thrust vector as opposing forces. With the inertial measurement unit, the vectors can be maintained by the attitude control system in parallel as well as keep the vectors opposing by using the velocity vector as input [17]. One of the great benefits of using gravity-turn descent is that an upright soft landing along with optimal fuel consumption is guaranteed [17]. As time goes on, a higher degree of landing accuracy is required for future missions.

The landing process can be described as the EDL profile - Entry, descent and Landing. These three phases constitute the landing mission of the space craft on the surface of the planetary surface. During the entry phase the craft enters the atmosphere and is propelled by thrusters to keep the proper trajectory as the atmosphere works against it. Once the lander slows down, the parachute is deployed and starts with the descent phase. The heat shields are dropped and the lander has visuals on its surroundings as it is being exposed to the atmosphere. The lander has to calculate the accurate landing trajectory in the limited amount of time for its travel between the atmosphere and the surface. In order to land safely, the parachute is cut off after further reducing its speed and the powered descent phase begins with the thrusters accurately guiding the lander to a safe landing spot [19].

RL algorithms are used during the descent and the powered descent phase. The input about the environment is through the different sensors and cameras around the lander which is used to accurately measure the current position and the trajectory of the lander. The lander uses the thrusters according to the policy calculated to pick out the best action for the lander in order to achieve pinpoint soft landing in the most fuel efficient way.

II. Literature

A. Reinforcement learning

Reinforcement learning is the process of learning what needs to be done given a particular situation observed. This involves mapping between the actions possible and the state the environment is currently in, thereby through a discovery process maximise the reward signal produced by the environment [3]. It can be formalized by the Markov decision process which can be described as the tuple (*S, A, P, R,* γ) where: [3]

State, *S* : Set of states which describe the different configurations the environment can be present in.

Action, *A* : Set of actions the learner or agent can take to interact with the environment thereby controlling the system state transitions.

Reward, *R* : The reward signal produced by the environment when transitioning from a previous old state to a new state which provides a feedback to the agent. The rewards can be defined as:

$$Totalrewards = \sum_{i=1}^{T} \gamma^{i-1} r^i$$

Transition function,*P* :The transition function describes the probability distribution over a set of possible transitions between the different states of the environment. For a Markovian process, the transition function for the current state and is dependent only on the previous state. It can be mathematically described as: $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$

Discount Factor , γ : the discount factor that is bounded between 0 and 1 that helps with the tradeoff between exploration and exploitation. [3]

The main sub-elements that defines the reinforcement learning problem are: [3]

Policy: It is the mapping between the action set A and the set of states perceived by the agent about the environment, thus defining the behaviour of the agent. It can take the form of a function, a lookup table or search processes which involves extensive computation. [3]

Reward signal: It defines the goal or the objective of the agent in the environment. A feedback provided by the environment to the agent to indicate the direction. The objective of the agent is to maximise the total rewards obtained by the environment thereby the agent learns by updating the policy based on the reward signal obtained. [3]

Value function: A predictive factor of how far the agent it to the goal or objective. The value of a state defines the total rewards the agent can be expected to accumulate towards reaching the objective or the end goal.

There are different ways reinforcement learning algorithms can be categorized. One such way is by the function being optimized for learning. Reinforcement learning algorithms can be classified into value function algorithms and policy gradient algorithms. With algorithms based on the value function, the agent learns to map between the state-action pair tuple and the total discounted future rewards obtained from beginning at that specific state with that particular action. It is also seen as optimizing the value function at each state. One of the main issues with this class of algorithms is that the actions are greedily chosen based on the values of each action computed at a state which limits the exploration aspect of the agent. This leads to the need to consider all the possible state-action combinations which could rise exponentially with an increase in the number of states and actions. Due to this reason, only discreet action spaces can be considered to implement this class of algorithms.

B. Policy gradient methods

Reinforcement learning techniques that optimizes the parameterized policy explicitly rather than improving the value function at each state. These techniques are needed especially when the policy is stochastic in nature rather than deterministic in nature. This means that different actions have specific probabilities at each state thereby giving it a better convergence since an arbitrary change in the estimated value of an action does not highly have an impact in the decision for that action [5]. The policy is optimized using gradient ascent algorithm. A general way to estimate the gradient is:

$$\hat{J} = \hat{E}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

where the stochastic policy is π_θ and the estimate advantage function at timestep *t* is $\hat{A}_t$. It calculates the expectation or an experimental average over a limited set of data where the algorithm computes the different samples as well as optimizes the algorithm in turns.

Policy gradient methods follow the general schema in order to maximise the objective $J(\theta)$ such that [3] $\theta_{t+1} = \theta_t + \alpha\nabla\hat{J}(\theta_t)$,Where $J(\theta_t)$ is a stochastic estimate which approximates the gradient performance measure with respect to θ_t, the policy parameter [3]. Thus the generic loss function can be derived from (1) to obtain:

$$L^{PG}(\theta) = \hat{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2}$$

C. Trust region Policy Optimization

Trust region Policy Optimization or TRPO algorithm is an important policy gradient algorithm as it introduces the concept of trust regions which help the updates be constrained with KL - divergence in order to avoid too much deviation of the policy. Regular policy gradient algorithms use the first order derivative for optimization which presumes the learning curve to be a flat surface. Thus the direction in which the policy is updates would be too steep which would lead to a complete deviation of the behaviour required for complex problems. With trust regions, the constraint updates would prevent this from occurring.

The Trust region Policy Optimization updates the policy by optimizing a surrogate objective. The policy gradient estimator is the gradient of the objective function constrained on the KL-divergence between the original policy and the updated policy [1].

Thus the objective function that is maximised subjected to the constraint is given by:

$$maximize_\theta\ \hat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{3}$$

$$subject\ to\ \hat{E}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta \tag{4}$$

Where theta old is the original policy prior to the update. By making linear approximation to the objective and quadratic approximation to the constraint, the problem can be solved approximately using the conjugate gradient algorithm. Instead of using a constraint, the problem can be formulated using a penalty instead thereby solving the unconstrained optimization problem [7].

$$maximize_\theta\ \hat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] - \beta KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \tag{5}$$

Where the Kullback-Leibler (KL) divergence is the KL() function. The gradients are obtained from the second order function which makes the calculations more complex and computationally expensive. The other issue faced by the method is maintaining a stable performance across different problems or a single problem with changing characteristics over time by choosing a single value of Beta [2].

D. Policy Proximal optimization

Policy Proximal optimization simplifies the objective function of TRPO while implementing a similar constraint. PPO algorithms have different techniques in order to implement the trust region function. The initial implementation of PPO is using a KL penalty which is modified to be much more adaptive. The later variation intoduces a surrogate objective:

$$J^{CLIP}(\theta) = E[min(r(\theta)\hat{A}_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s,a)] \tag{6}$$

where:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \tag{7}$$

is the probability ratio between the original policy and the updated policy.

The term $clip(r(\theta), 1-\epsilon, 1+\epsilon)$, maintains the ratio between 1-e and 1+e thus clipping the value of the ratio beyond the terms thereby avoiding the updates to be extremely large thus preventing the convergence to collapse. The objective function thus decides to use the minimum of either the regular policy iteration or the clipped function thus keeping it with the spirit of trust regions [2].

PPO algorithms are currently the most popular as it is currently seen is the best performing algorithm in most of the problems and has simpler computations due to the fact that only the first derivative needs to be considered. The better performance is mainly due to the fact that is has better sample efficiency compared to the previous algorithms.

III. Methodology

A. Environment

It is a collection of Reinforcement Learning environments' benchmarks ranging from Atari games to simplistic robotic simulations used to test the different RL algorithms making is accessible and convenient to use. It contains a variety of tasks called environments which grows as time goes on. The convenience of these environments is that they are accessible with a common interface. The platform provides an abstraction of the environment which provides flexibility on the different types of agents that can be used. LunarLander-v2 environment was used to depict the lander on a planetary surface using the BOX2D simulator. [21]

Observation Space: The observation space is the set of attributes that are available to the agent from the environment. The environment has 8 variables associated to the state space:

- x coordinate of the lander
- y coordinate of the lander
- v_x the horizontal velocity
- v_y the horizontal velocity
- θ the orientation in space
- v_θ the angular velocity
- Left leg touching the ground (Boolean)
- Right leg touching the ground (Boolean)

The coordinate system of the environment is such that the landing pad is located at the origin or the (0,0) mark with all the coordinates relative to that position. Thus the coordinates at the bottom or left of the pad is negative whereas the positions on the right or top of the landing pad is positive.

Action Space: The action space for the environment is discreet with four available actions: Fire left orientation engine, fire right orientation engine, fire main engine and do nothing. The main engine controls the altitude of the lander while the left and right engines controls the torque of the lander which affects its orientation and stabilization.

Reward: The rewards are in the range of 100 to 140 points which is simply obtained by moving from the top to the landing pad without any velocity. Reward points are deducted for the distance away from the landing pad. A reward of additional 10 point is given for contact of each leg on the ground. The episode ends either when the lander has landed safely or has crashed awarding 100 or deducting 100 respectively. 0.3 points are deducted from the reward when the main engine is fired at each frame.

The goal of the environment is that the lander has to land on the landing pad provided with both the legs of the lander directly in contact with the pad. [22]

B. Phasic Policy Gradient

In PPO methods, the value and policy networks share parameters which allows the sharing of learnt features with each other. The disadvantage of sharing parameters is that it becomes difficult to balance the policy and value function objectives. If the method optimizes both the objectives together, a relative weight would have to be assigned to it thereby optimizing one objective would have an opposite impact to the other. The other issue faced by sharing a network is the requirement of both the value and policy function objectives to be trained by using the same data, thus maintaining the same sample reuse for the network which causes an undesirable and artificial restriction. [7]

With the PPG methods, the feature sharing is preserved as well as the training of both the objective and value functions are decoupled. Thus it operated by using two alternating phases:

- The policy phase where in which the policy function objective is optimized.
- The auxiliary phase where the useful features from the value function is distilled to the policy network. This improves the training of the policy.

Thus when the interface between the value and policy function objectives are mitigated, PPG improves the sample efficiency significantly while still sharing the representations as well as optimizing both the functions with the same level of sample reuse. [7]

In the Policy phase, the policy is optimized using the same objective as the PPO objective. Along with optimizing the policy function, the value function objective is also optimized:

$$L^{value} = \hat{E}_t[\frac{1}{2}(V_{\theta_v}(s_t) \; - \; \hat{V}_t^{targ})^2] \tag{8}$$

$\hat{V}_t^{targ}$ denotes the value function targets. A generalized advantage estimation function derives the values of the advantage $\hat{A}$ and $\hat{V}_t^{targ}$. [7]

The generalized advantage estimation function is a combination of the discounted sum of Temporal-Differencing residuals $\delta_t^\pi = r_t + \gamma V(s_{t+1}) - V(s_t)$ and the discounted advantage for k-steps of the agent, $\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^\pi$. It can be defined as the average of these estimators [29]:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma,\lambda)^l \delta_{t+l}^\pi \tag{9}$$

After the policy phase, the network is further updated and optimized by the auxiliary phase. The loss function is defined as:

$$L^{joint} = L^{aux} \; \beta_{clone} \hat{E}[KL[\pi_{\theta_o ld}(.|s_t), \pi_\theta(.|s_t)]] \tag{10}$$

where $\pi_{\theta_o ld}$ is the updated policy from the policy phase before entering into the auxiliary phase.t can be seen that the auxillary objective is optimized while the original policy is preserved. This is achieved with the hyperparameter β_{clone} that controls the trade-off.

C. Network setup

The implementation follows the actor critic model with a network for each component. The Actor network has two hidden layers with the Tanh activation function after the input layer layers as well as the first hidden layer whereas the second hidden layer has a ReLU activation function. The actor network acts a parametrised policy which selects the action. It is used to for evaluating the current policy and obtaining the probabilities of the different actions at the current state. The actor network also provides with an auxiliary value for the current values that is used for training in the auxiliary phase. It can be described as two different heads, an action head that provides the action probabilities at each state and a value head which evaluates an auxiliary value at that state.

The critic network has three layers including the input and output layer along with Tanh activation function. The critic network evaluates the state of the environment after the agent takes the action suggested by the actor network. Thus the critic network works as a value evaluation function which provides

Algorithm 1 PPG Algorithm [7]

for $Phase = 1, 2, ...$ **do** Initialize empty buffer B
  for $Iteration = 1, 2...., N_\pi$ **do** ▷ Policy Phase
    Perform rollouts under current policy π
    Compute Value function target $\hat{V}_t^{targ}$
    for $epoch = 1, 2...., E_\pi$ **do** ▷ Policy Epochs
      Optimize $L^{clip} + \beta_S S[\pi]$ wrt θ_π
    end for
    for $epoch = 1, 2...., E_V$ **do** ▷ Value Epochs
      Optimize L^{value} wrt θ_V
    end for
    Add all $(s_t, \hat{V}_t^{targ})$ to B
  end for
  Compute and store current policy $\pi_{\theta_{old}}(.|s_t)$ for all states s_t in B
  for $epoch = 1, 2...., E_{aux}$ **do** ▷ Auxiliary Phase
    Optimize L^{joint} wrt θ_π, on all data in B
    Optimize L^{value} wrt θ_V, on all data in B
  end for
end for

feedback to the actor network of how well the current state is as evaluate the future states.

The actor-critic architecture is prominently used especially for online learning. It combines the advantages of the actor network being parameterized which helps in determining the action in the continuous action space without the need of improving the value function as well as the critic network which provides the policy network with low variance information of the performance of the policy. This provides better convergence to the learning compared to policy only or value function estimation methods. The network constructed implements policy gradient techniques, i.e. the phasic policy gradient algorithm on actor critic structure. The implementation considers the time steps to be discreet rather than continuous [25].

The activation functions used are the hyperbolic tangent function (TanH) for the hidden layers and the Rectified Linear Unit function (ReLU) after the output layer. The primary importance of the activation function in an artificial neural network is to convert the input signal to an output suitable for the need whether it is to transfer the signal to the hidden layer or as the final output. The product between weights and input signal for each node in the input layer is calculated and all the products are added together to produce the output of a single node of the next layer. Without the activation function or with a linear activation function, the network would function as a regular linear regression model since the input signal and the output signal would be very similar. TanH and ReLU are non linear activation functions which rectifies the errors that come across along with the input signal which would have non-linear characteristics thereby allowing the network to learn the different and complicated relationships between the data points [26].

Hyperbolic tangent function squashes the input signal into the range between -1 and 1 using the formula:

$$f_2(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \tag{11}$$

It deals with the vanishing gradient problem to a certain extent by keeping the near-zero values the same unlike the sigmoid function which maps the values to either 0 or 1 making it prone to the problem. The function is differentiable with the derivative of $\tanh x$ being $1 - \tanh^2 x$ making it easier for backpropagation by reusing the values [26].

The rectified linear activation function can be described as Max$\{0$, input$\}$. Thus making it linear for the positive part of the domain and non liner for the negative part of the domain. ReL is an important activation function due to its simplicity and effectiveness in training the neural network. The derivative of the function is simply the slope of the line which is easy to calculate. For negative values the derivative obtained would be 0 and the derivative of the positive values will always be 1. With the TanH and the ReLU activation, probabilities for each action can be obtained [27].

An optimization function helps in learning the right weights for the network nodes with the help of different parameters such as the learning rate in order to minimize the losses created by the agent. The optimization algorithm used is the Adam optimizer which stands for adaptive moment estimation. Adam is a very effective optimization algorithm especially for deep neural networks and can be seen as an extension of the stochastic gradient descent. It is a combination of the Adaptive gradient algorithm (AdaGrad) and the Root Mean Square Propagation (RMSProp). The advantage of the both the algorithms is that the learning rate is maintained when the performance of the agent improves even with sparse gradients as well as adapting the learning rates based on how quickly the weights change by calculating the average of the most recent gradients giving the algorithm the flexibility needed for online learning. Thus the only requirement for the algorithm The parameters of the optimizer are the learning rate or alpha that refers to the step size for each weight update and beta1 and beta2 to control the decay rate of the gradient which is computed as the exponential moving average for the first and second moments respectively. Another parameter used is the epsilon which is a very small number that helps prevent any division by zero which may occur. The values set for each of the parameters is shown in the table below.[41]

In order to gain sample efficiency, buffers are used to store a set of previous states and actions that is used in the learning algorithm. The samples are shuffled in order to decrease the bias during the learning. There are two buffers used, the main memory buffer used in the policy phase training and the auxiliary buffer used during the auxiliary phase. The main memory buffer stores the state, action and reward obtained for a set number of episodes. During the policy phase training, the values of the states are calculated and then stored in the auxiliary buffer along with the current state and reward obtained at that state.

The actions are stored as a categorical distribution. During training, the probability values are calculated and stored as a distribution. The distribution is sampled to obtain the best action for the particular state.

IV. Experiments and Analysis

The implementation is done using pytorch. The experimental analysis includes comparing the impact of different activation functions and optimizers, varying the number of layers and tunning the different hyperparameters. Varying the different aspects in the learning algorithm helps to ensure that the best results are obtained and thus the agent learns well.

A. Activation Function

As previously mentioned, the main activation functions used within the network is the Tanh and the ReLU activation. Keeping the optimizer as Adam as a constant and with the hyperparameters as mentioned in table, the network is modified with different combinations of the activation functions. Constructing the network with only TanH activation between layers, makes the learning process quick but poor. The agent learns to tilt towards one side and crash land. Whereas using only ReLU as the activation function between the layers, the network faces the issue of vanishing/exploding gradients which throws an error and the learning cannot continue. The learning curve tends to be better, thereby the agent works better when both the activation functions are combined between the layers.

TABLE I
USING TANH ACTIVATION FUNCTION

Hyperparameter	Value
Number of Episodes	5000
Learning Rate	0.0005
λ	0.95
γ	0.99
Policy Phase Iterations	1
Auxiliary Phase Iterations	6

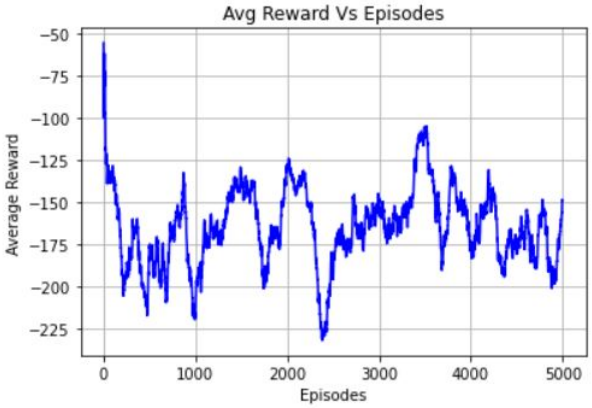

Fig. 1. Average rewards obtained while training using the TanH activation only

B. Optimizer

Here the Stochastic gradient descent and the Adam optimizer are compared. Stochastic gradient descent algorithm optimizes the weights of the network after each sample and performs the vanilla gradient decent. The learning converges very slowly and the performance is not adequate due to the fact that there are high probabilities that the learning gets trapped around the local optima making it difficult to obtain the true optimized value. It is observed that Adam optimizer performs much better in relation to the Stochastic gradient descent optimizer. Adam optimizer is very adaptable as it uses a wide range of learning rates for each of the parameter used by the network. This flexibility allows the optimizer to obtain an optimum value and avoid the values lingering around local areas.

TABLE II
USING SGD OPTIMIZER

Hyperparameter	Value
Number of Episodes	5000
Learning Rate	0.0005
λ	0.95
γ	0.99
Policy Phase Iterations	1
Auxiliary Phase Iterations	6

Fig. 2. Average rewards obtained while training using the SGD Optimization function

TABLE III
USING ADAM OPTIMIZER

Hyperparameter	Value
Number of Episodes	5000
Learning Rate	0.0005
λ	0.95
γ	0.99
Policy Phase Iterations	1
Auxiliary Phase Iterations	6

C. Hyperparameters

Another variation experimented on was the number of iterations needed for the phases. It is seen that the performance

is optimal when the number of iterations used for the policy phase is lower than that of the auxiliary phase. The higher number of iterations for each, shows a degradation in performance. Hence the number of iterations for both the phases have to be kept at a minimum. This is due to the fact that the increase in number of samples in the buffers biases the learning.

TABLE IV
TUINING THE HYPERPARAMETERS

Hyperparameter	Value
Number of Episodes	5000
Learning Rate	0.0005
λ	0.90
γ	0.95
Policy Phase Iterations	10
Auxiliary Phase Iterations	20

Fig. 3. Average rewards obtained while training with tuned hyperparameters

V. RESULT

Combining the best of the result obtained from the different experimentation explained above the agent works well enough to land on the surface with less chances of crashing. The combination of the TanH and ReLU activation function and the Adam optimizer is used as well as the tuned hyperparameter values. Table 5 lists the settings and figure 4 shows the average reward obtained during training.

TABLE V
HYPERPARAMETERS OF THE WORKING AGENT

Hyperparameter	Value
Number of Episodes	5000
Learning Rate	0.0005
λ	0.90
γ	0.95
Policy Phase Iterations	1
Auxiliary Phase Iterations	6

VI. CONCLUSION

This implementation is basic and rudimentary and has lots of room for improvement. The current system works on a 3 DOF lander with infinite fuel. For experimental purpose, this

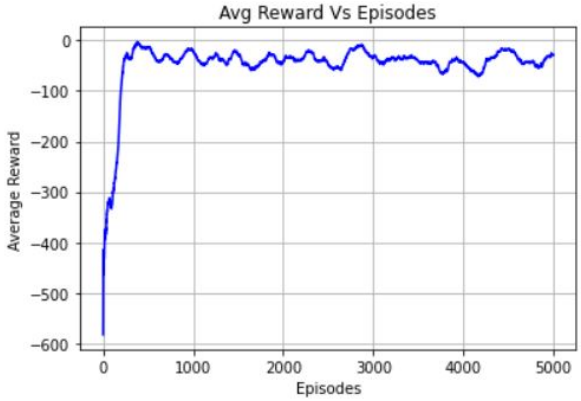

Fig. 4. Average rewards of the agent

environment is sufficient but is far away from the real scenario. Any space craft will have more than 3 degrees of freedom and thus the additional degrees of freedom has to be considered. The fuel in the current open AI environment is infinite and is no practical no implement algorithms that needs to be used in the real world. Hence the algorithm needs to be modified to consider fuel efficiency. The current agent does not learn well and thus needs more hyper parameter tunning.

ACKNOWLEDGMENT

I would like to thank Dr. Angadh Nanjungud for his motivation and support throughout this project. I would also like to thank the University for providing resources crucial to the project.

REFERENCES

[1] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, P. Abbeel, “Trust Region Policy Optimization,”
[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms”. In: CoRR abs/1707.06347 (2017). arXiv: 1707.06347
[3] R. S. Sutton and A. G. Barto. Introduction to reinforcement learning MIT Press, 1998.
[4] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau (2018), “An Introduction to Deep Reinforcement Learning”, Foundations and Trends in Machine Learning: Vol. 11, No. 3-4. DOI: 10.1561/2200000071
[5] Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour, “Policy Gradient Methods for Reinforcement Learning with Function Approximation ,” ATT Labs - Research, 180 Park Avenue, Florham Park, NJ 07932.
[6] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In Advances in neural information processing systems, pages 5279-5288, 2017.
[7] Karl W Cobbe, Jacob Hilton, Oleg Klimov, John Schulman. Phasic Policy Gradient. Proceedings of the 38th International Conference on Machine Learning, PMLR 139:2020-2027, 2021
[8] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, Moritz Hardt. Revisiting Design Choices in Proximal Policy Optimization. arXiv:2009.10897
[9] B Gaudet, R Furfaro. Robust spacecraft hovering near small bodies in environments with unknown dynamics using reinforcement learning. AIAA/AAS Astrodynamics Specialist Conference, 2012
[10] Izzo, D., Märtens, M. Pan, B. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. Astrodyn 3, 287–299 (2019). https://doi.org/10.1007/s42064-018-0053-6

[11] Brian Gaudet, Richard Linares, Roberto Furfaro. Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing.
[12] Andrea Scorsoglio, Andrea DAmbrosio, Luca Ghilardi, Roberto Furfaro, Brian Gaudet, Richard Linaresk, Fabio Curti. SAFE LUNAR LANDING VIA IMAGES: A REINFORCEMENT META-LEARNING APPLICATION TO AUTONOMOUS HAZARD AVOIDANCE AND LANDING
[13] Logsdon, J. M.. "space exploration." Encyclopedia Britannica, July 30, 2021. https://www.britannica.com/science/space-exploration.
[14] CHAD R. FROST, Challenges and Opportunities for Autonomous Systems in Space, NASA Ames Research Center.
[15] Andrew Harris,Thibaud Teil,Hanspeter Schaub. SPACECRAFT DECISION-MAKING AUTONOMY USING DEEP REINFORCEMENT LEARNING.
[16] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), 2011, pp. 1143-1146, doi: 10.1109/MEC.2011.6025669.
[17] Ibrahim M. Mehedi, Takashi Kubota, Advanced Descent Scheme for Lunar Landing, IFAC Proceedings Volumes, Volume 43, Issue 15, 2010, Pages 166-171, ISSN 1474-6670, ISBN 9783902661968, https://doi.org/10.3182/20100906-5-JP-2022.00029.
[18] Jie Dong, Zezhou Sun, Wei Rao, Yang Jia, Linzhi Meng, Chuang Wang, Baichao Chen, MISSION PROFILE AND DESIGN CHALLENGES FOR MARS LANDING EXPLORATION
[19] mars.nasa.gov/mars2020/timeline/landing/entry-descent-landing/
[20] B. Gaudet and R. Furfaro, "Adaptive pinpoint and fuel efficient mars landing using reinforcement learning," in IEEE/CAA Journal of Automatica Sinica, vol. 1, no. 4, pp. 397-411, Oct. 2014, doi: 10.1109/JAS.2014.7004667.
[21] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba. OpenAI Gym
[22] Soham Gadgil, Yunfeng Xin, Chengzhe Xu. Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning.
[23] M. M. Lau and K. H. Lim, "Investigation of activation functions in deep belief network," 2017 2nd International Conference on Control and Robotics Engineering (ICCRE), 2017, pp. 201-206, doi: 10.1109/ICCRE.2017.7935070.
[24] I. Grondman, L. Busoniu, G. A. D. Lopes and R. Babuska, "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1291-1307, Nov. 2012, doi: 10.1109/TSMCC.2012.2218595.
[25] Vijay R. Konda, John N. Tsitsiklis. Actor-Critic Algorithms.
[26] Siddharth Sharma, Simone Sharma, ACTIVATION FUNCTIONS IN NEURAL NETWORKS
[27] Vinod Nair, Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines.
[28] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980
[29] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. arXiv:1506.02438